min max

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <climits>
using namespace std;

void min_reduction(vector<int>& arr) {
int min_value = INT_MAX;
#pragma omp parallel for reduction(min: min_value)
for (int i = 0; i < arr.size(); i++) {
if (arr[i] < min_value) {
min_value = arr[i];
}
}
cout << "Minimum value: " << min_value << endl;
}
void max_reduction(vector<int>& arr) {
int max_value = INT_MIN;
#pragma omp parallel for reduction(max: max_value)
for (int i = 0; i < arr.size(); i++) {
if (arr[i] > max_value) {
max_value = arr[i];
}
}
cout << "Maximum value: " << max_value << endl;
}
void sum_reduction(vector<int>& arr) {
```

```
int sum = 0;
#pragma omp parallel for reduction(+: sum)
for (int i = 0; i < arr.size(); i++) {
sum += arr[i];
}
cout << "Sum: " << sum << endl;
}
void average_reduction(vector<int>& arr) {
int sum = 0;
#pragma omp parallel for reduction(+: sum)
for (int i = 0; i < arr.size(); i++) {
sum += arr[i];
}
cout << "Average: " << (double)sum / arr.size() << endl;
}
int main() {
vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
min_reduction(arr);
max_reduction(arr);
sum_reduction(arr);
average_reduction(arr);
}
```

---

bubble sort

```
#include <iostream>
#include <vector>
```

```
#include <omp.h>
using namespace std;

void bubble_sort_odd_even(vector<int>& arr) {
bool isSorted = false;
while (!isSorted) {
isSorted = true;
#pragma omp parallel for
for (int i = 0; i < arr.size() - 1; i += 2) {
if (arr[i] > arr[i + 1]) {
swap(arr[i], arr[i + 1]);
isSorted = false;
}
}
#pragma omp parallel for
for (int i = 1; i < arr.size() - 1; i += 2) {
if (arr[i] > arr[i + 1]) {
swap(arr[i], arr[i + 1]);
isSorted = false;
}
}
}
}

int main() {
vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
double start, end;
// Measure performance of parallel bubble sort using odd-
//even transposition
```

```
start = omp_get_wtime();
bubble_sort_odd_even(arr);
end = omp_get_wtime();
cout << "Parallel bubble sort using odd-even transposition time: " << end - start << endl;
}
```

---

merge sort

```
include <iostream>
#include <vector>
#include <omp.h>
using namespace std;

void merge(vector<int>& arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    vector<int> L(n1), R(n2);
    for (i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }
    for (j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
```

```
            if (L[i] <= R[j]) {
                arr[k++] = L[i++];
            } else {
                arr[k++] = R[j++];
            }
        }
}
void merge_sort(vector<int>& arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
#pragma omp task
            merge_sort(arr, l, m);
#pragma omp task
            merge_sort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
}
void parallel_merge_sort(vector<int>& arr) {
#pragma omp parallel
        {
#pragma omp single
            merge_sort(arr, 0, arr.size() - 1);
        }
}
int main() {
        vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
        double start, end;
        // Measure performance of sequential merge sort
        start = omp_get_wtime();
```

```
    merge_sort(arr, 0, arr.size() - 1);
    end = omp_get_wtime();
    cout << "Sequential merge sort time: " << end - start <<endl;
    // Measure performance of parallel merge sort
    arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
    start = omp_get_wtime();
    parallel_merge_sort(arr);
    end = omp_get_wtime();
    cout << "Parallel merge sort time: " << end - start <<endl;
    }
```

---

bfs

```
#include <iostream>
#include <vector>
#include <queue>
#include <omp.h>
using namespace std;

const int MAXN = 1e5;
vector<int> adj[MAXN + 5]; // adjacency list
bool visited[MAXN + 5];    // mark visited nodes

void bfs(int start_node) {
   queue<int> queue;
   queue.push(start_node);

   while (!queue.empty()) {
```

```
        int current_node = queue.front();
        queue.pop();

        if (!visited[current_node]) {
            visited[current_node] = true;
            cout << "Visited node: " << current_node << endl;

            #pragma omp parallel for
            for (int i = 0; i < adj[current_node].size(); i++) {
                int next_node = adj[current_node][i];
                if (!visited[next_node]) {
                    #pragma omp critical
                    queue.push(next_node);
                }
            }
        }
    }
}

int main() {
    cout << "Please enter the number of nodes: ";
    int n; // number of nodes
    cin >> n;

    cout << "Please enter the number of edges: ";
    int m; // number of edges
    cin >> m;

    for (int i = 1; i <= m; i++) {
```

```
        cout << "Enter edge " << i << ": ";
        int u, v; // edge between u and v
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    cout << "Enter the starting node of BFS: ";
    int start_node; // start node of BFS
    cin >> start_node;

    bfs(start_node);

    return 0;
}
```

---

dfs

```
#include <iostream>
#include <vector>
#include <stack>
#include <omp.h>
using namespace std;

const int MAXN = 1e5;

vector<int> adj[MAXN + 5]; // adjacency list
bool visited[MAXN + 5];    // mark visited nodes
```

```
void dfs(int start_node) {
    stack<int> stack;
    stack.push(start_node);

    while (!stack.empty()) {
        int current_node = stack.top();
        stack.pop();

        if (!visited[current_node]) {
            visited[current_node] = true;
            cout << "Visited node: " << current_node << endl;

            #pragma omp parallel for
            for (int i = 0; i < adj[current_node].size(); i++) {
                int next_node = adj[current_node][i];
                if (!visited[next_node]) {
                    #pragma omp critical
                    stack.push(next_node);
                }
            }
        }
    }
}

int main() {
    cout << "Please enter the number of nodes: ";
    int n; // number of nodes
    cin >> n;
```

```
    cout << "Please enter the number of edges: ";
    int m; // number of edges
    cin >> m;

    for (int i = 1; i <= m; i++) {
        cout << "Enter edge " << i << ": ";
        int u, v; // edge between u and v
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    cout << "Enter the starting node of DFS: ";
    int start_node; // start node of DFS
    cin >> start_node;

    dfs(start_node);

    return 0;
}
```

```
#Assignment 1 : Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
from sklearn import preprocessing
```

```
(X_train,y_train),(X_test,y_test)=keras.datasets.boston_housing.load_data()
```

```
print("Training data shape:",X_train.shape)
```

```
print("Testing data shape:",X_test.shape)
```

```
print("Train output data shape:",y_train.shape)

print("Actual Test  output data shape:",y_test.shape)
```

```
X_train[0]
```

```
y_train[0]
```

```
y_train
```

```
## Normalize the data
```

```
X_train=preprocessing.normalize(X_train)
```

```
X_train
```

```
X_train[0]
```

```
X_train[1]
```

```
X_test=preprocessing.normalize(X_test)
```

```
X_test
```

```
X_test[0]
```

```
y_test[0]
```

```
## Model Building
X_train[0].shape
```

```
model=Sequential()
model.add(Dense(128,activation='relu',input_shape= X_train[0].shape))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(1))
```

```
model.summary()
```

```
model.compile(loss='mse',optimizer='rmsprop',metrics=['mae'])
```

```
history=
model.fit(X_train,y_train,epochs=100,batch_size=1,verbose=1,validation_dat
a=(X_test,y_test))
```

```
test_input=[(0.0024119 , 0.        , 0.01592969, 0.        , 0.00105285,
       0.01201967, 0.17945359, 0.00778265, 0.00782786, 0.6007879 ,
       0.04109624, 0.77671895, 0.03663436)]
print("Actual output:15.2")
print("Predicted Output:",model.predict(test_input))
```

```
test_input=[(4.07923050e-05, 1.54587284e-01, 3.80378407e-03,
0.00000000e+00,
       7.77620881e-04, 1.42595058e-02, 2.94184285e-02, 1.17486336e-02,
       3.74757051e-03, 6.52077269e-01, 2.75446433e-02, 7.40857215e-01,
       5.82747215e-03)]
print("Actual output:42.3")
print("Predicted Output:",model.predict(test_input))
```

```
y_pred=model.predict(X_test)
```

```
y_pred
```

```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```